第01天 java面向对象

今日内容介绍

* 知识回顾
* static静态关键字
* 代码块

# 知识回顾

## 方法的回顾

### 案例代码一:

package com.itheima\_01;

/\*

\* 需求：定义一个方法求两个数的和,并在主方法中调用

\*

\* 方法：类中的一段具有特定功能的程序，提高了代码的复用性和可维护性

\* 定义格式：

\* public static 返回值类型（没有返回值写void） 方法名(参数类型 参数名,参数类型 参数名2) {//形参

\* 方法体;

\* }

\* 调用方式：

\* 有明确返回值类型：

\* 赋值调用，将方法的返回值赋值给一个变量

\* 输出调用，使用输出语句直接输出方法的返回值

\* 直接调用，没法获取方法的返回值

\* 没有明确返回值类型：

\* 直接调用

\* 方法重载：在一个类中有多个重名的方法，这些方法参数不同，和返回值无关

\*

\* 注意：

\* 形参：方法声明的变量，只能是变量，接收方法调用时传递进来的数据

\* 实参：调用方法时传递的数据，可以是常量也可以是变量

\*

\*/

public class MethoDemo {

public static void main(String[] args) {

//赋值调用

//int sum = sum(10,20);//实参

//System.out.println(sum);

//输出调用

int a = 10;

int b = 20;

System.out.println(sum(a,b));

}

public static int sum(int a,int b) {

/\* //使用变量接收求和结果并返回

int sum = a + b;

return sum;\*/

//直接返回求和结果

return a + b;

}

}

## 数组的回顾

### 案例代码二:

package com.itheima\_02;

/\*

\* 需求：定义一个元素类型为int的数组，遍历数组并求和

\*

\* 数组：用于存储多个元素的一种容器

\* 数组的特点:

\* 元素类型必须一致

\* 元素有整数索引

\* 一旦定义好长度则无法改变

\* 可以存储基本数据类型

\* 也可以存储引用数据类型

\* 定义格式：

\* 动态初始化

\* 元素类型[] 数组名 = new 元素类型[10];

\* 静态初始化

\* 元素类型[] 数组名 = {元素1,元素2,元素3};

\* 元素类型[] 数组名 = new 元素类型[]{元素1,元素2,元素3};

\*

\*/

public class ArrayDemo {

public static void main(String[] args) {

//使用静态初始化定义数组

int[] arr = {1,2,3,4,5};

//定义一个变量用于存储求和结果

int sum = 0;

//遍历数组

for(int x = 0;x < arr.length;x++) {

sum += arr[x];

}

System.out.println(sum);

}

}

## 标准类定义和使用回顾

### 案例代码三:

package com.itheima\_03;

/\*

\* 定义一个标准的学生类，在主方法中创建对象并调用

\* 姓名，年龄，性别3个成员变量

\* 无参，有参两个构造方法

\* 为每个成员变量定义getter/setter方法

\* 定义一个show方法，输出成员变量

\*/

public class Student {

private String name;//姓名

private int age;//年龄

private String gender;//性别

/\*//无参构造

public Student() {}

//有参构造

public Student(String name,int age,String gender) {

this.name = name;

this.age = age;

this.gender = gender;

}

//name

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

//age

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

//gender

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}\*/

//show：用于输出所有的成员变量

public void show() {

System.out.println(name + "," + age + "," + gender);

}

public Student() {

super();

// TODO Auto-generated constructor stub

}

public Student(String name, int age, String gender) {

super();

this.name = name;

this.age = age;

this.gender = gender;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

}

package com.itheima\_03;

public class StudentTest {

public static void main(String[] args) {

//创建学生对象

Student s = new Student();

//为成员变量进行赋值

s.setName("张三");

s.setAge(18);

s.setGender("男");

s.show();

System.out.println("----------");

Student s2 = new Student("李四",20,"其他");

//s2.show();

System.out.println(s2.getName());

}

}

# static静态关键字

## 静态的概述

当在定义类的时候，类中都会有相应的属性和方法。而属性和方法都是通过创建本类对象调用的。当在调用对象的某个方法时，这个方法没有访问到对象的特有数据时，方法创建这个对象有些多余。可是不创建对象，方法又调用不了，这时就会想，那么我们能不能不创建对象，就可以调用方法呢？

可以的，我们可以通过static关键字来实现。static它是静态修饰符，一般用来修饰类中的成员。

## 静态的特点

A: 被static修饰的成员变量属于类，不属于这个类的某个对象。（也就是说，多个对象在访问或修改static修饰的成员变量时，其中一个对象将static成员变量值进行了修改，其他对象中的static成员变量值跟着改变，即多个对象共享同一个static成员变量）

B: 被static修饰的成员可以并且建议通过类名直接访问

访问静态成员的格式:

类名.静态成员变量名

类名.静态成员方法名(参数)

C: 静态的加载优先于对象,随着类的加载而加载

### 案例代码四

package com.itheima\_01;

/\*

\* static:是一个关键字，用于修饰成员变量和成员方法

\* static的特点：

\* 被所有的对象所共享

\* 可以使用类名调用

\* 静态的加载优先于对象

\* 随着类的加载而加载

\*

\*/

public class StaticDemo {

public static void main(String[] args) {

Person.graduateFrom = "传智学院";

Person p = new Person();

p.name = "小苍同学";

p.age = 18;

//p.graduateFrom = "传智学院";

p.speak();

Person p2 = new Person();

p2.name = "小波同学";

p2.age = 20;

//p2.graduateFrom = "传智学院";

p2.speak();

}

}

class Person {

String name;

int age;

static String graduateFrom;//毕业院校

public void speak() {

System.out.println(name + "---" + graduateFrom);

}

}

## 静态的注意事项

A:静态成员只能直接访问静态成员

B:非静态成员既可以访问非静态成员也可以访问静态成员

### 案例代码五

package com.itheima\_01;

/\*

\* static的注意事项：

\* 静态方法：

\* 可以调用静态的成员变量

\* 可以调用静态的成员方法

\* 不可以调用非静态成员变量

\* 不可以调用非静态成员方法

\* 静态方法只能调用静态的成员

\* 非静态方法：

\* 可以调用静态的成员变量

\* 可以调用静态的成员方法

\* 可以调用非静态的成员变量

\* 可以调用非静态的成员方法

\*

\* 静态的方法中是否有this这个对象？没有的

\*

\*

\*/

public class StaticDemo2 {

public static void main(String[] args) {

Student.graduateFrom = "传智学院";

Student.study();

}

}

class Student {

String name;

int age;

static String graduateFrom;//毕业院校

public static void study() {

///System.out.println(graduateFrom);

//sleep();

//System.out.println(name);

//eat();

}

public static void sleep() {

System.out.println("sleep");

}

public void eat() {

System.out.println("eat");

System.out.println(graduateFrom);

sleep();

}

}

## 静态的优缺点

A:静态优点:

对对象的共享数据提供单独空间的存储，节省空间，没有必要每一个对象都存储一份

可以直接被类名调用,不用在堆内存创建对象

静态成员可以通过类名直接访问,相对创建对象访问成员方便

B:静态弊端:

访问出现局限性。（静态虽好，但只能访问静态）

## 静态应用

### Math类使用

A:Math 类包含用于执行基本数学运算的方法。数学操作常用的类。

B:Math类的构造方法被private,无法创建对象,也就无法通过对象来访问Math类中的成员

C:Math类中所有的成员都被静态修饰,因此我们可以直接通过类名访问

#### 案例代码三:

package com.itheima\_02;

public class MathDemo {

public static void main(String[] args) {

//Math:包含了一些基本的数学运算方法

//static double PI

//System.out.println(Math.PI);

//static double abs(double a) :返回绝对值

//System.out.println(Math.abs(15));

//System.out.println(Math.abs(-10));

//static double ceil(double a) 天花板 向上取整

//System.out.println(Math.ceil(1.2));

//System.out.println(Math.ceil(1.6));

//static double floor(double a) 地板 向下取整

//System.out.println(Math.floor(1.2));

//System.out.println(Math.floor(1.6));

//static long round(double a) ：四舍五入

//System.out.println(Math.round(1.2));

//System.out.println(Math.round(1.6));

//static double max(double a, double b)

//System.out.println(Math.max(3, 4));

//static double pow(double a, double b) :返回第一个参数的第二个参数次幂

//System.out.println(Math.pow(3, 2));

//static double random() :返回一个随机数，大于零且小于一

System.out.println(Math.random());

}

}

### 自定义工具类

A:需求:自定义一个专门对数组操作的工具类,具有的功能如下

1.定义一个方法,该方法可以返回数组中最大元素

2.定义一个方法,该方法根据指定的值去数组中查找是否存在该值

存在,返回该值在数组中的索引

不存在,返回-1

#### 案例代码四:

package com.itheima\_03;

public class MyArrays {

private MyArrays() {}

/\*

\* 返回数组中最大的元素

\*

\*/

public static int getMax(int[] arr) {

int max = 0;//参照物

//遍历数组

for(int x = 0;x < arr.length;x++) {

if(arr[x] > max) {

max = arr[x];//替换参照物

}

}

return max;

}

/\*

\* 返回数组中指定参数的索引

\*

\*/

public static int getIndex(int[] arr,int a) {

//遍历数组

for(int x = 0;x < arr.length;x++) {

if(arr[x] == a) {

return x;

}

}

return -1;//如果查不到制定的参数，则返回-1

}

}

package com.itheima\_03;

public class MyArraysDemo {

public static void main(String[] args) {

int[] arr = {3,5,8,10,1};

int max = MyArrays.getMax(arr);

System.out.println(max);

int index = MyArrays.getIndex(arr, 8);

System.out.println(index);

}

}

## 类变量与实例变量辨析

A:类变量:其实就是静态变量

定义位置:定义在类中方法外

所在内存区域:方法区

生命周期:随着类的加载而加载

特点:无论创建多少对象,类变量仅在方法区中,并且只有一份

B:实例变量:其实就是非静态变量

定义位置:定义在类中方法外

所在内存区域:堆

生命周期:随着对象的创建而加载

特点:每创建一个对象,堆中的对象中就有一份实例变量

# 代码块

## 局部代码块

局部代码块是定义在方法或语句中

### 案例代码六:

public class BlockDemo {

public static void main(String[] args) {

//局部代码块：存在于方法中，控制变量的生命周期（作用域）

{

for(int x = 0;x < 10;x++) {

System.out.println("我爱Java");

}

int num = 10;

}

//System.out.println(num);//无法访问num,超出num的作用域范围

}

}

## 构造代码块

构造代码块是定义在类中成员位置的代码块

### 案例代码七:

package com.itheima\_04;

class Teacher {

String name;

int age;

{

for(int x = 0;x < 10;x++) {

System.out.println("我爱Java");

}

System.out.println("我爱Java");

}

public Teacher() {

System.out.println("我是无参空构造");

}

public Teacher(String name,int age) {

System.out.println("我是有参构造");

this.name = name;

this.age = age;

}

}

## 静态代码块

A:静态代码块是定义在成员位置，使用static修饰的代码块

### 案例代码八:

class Teacher {

String name;

int age;

//静态代码块：随着类的加载而加载，只加载一次，加载类时需要做的一些初始化，比如加载驱动

static {

System.out.println("我爱Java");

}

public Teacher() {

System.out.println("我是无参空构造");

}

public Teacher(String name,int age) {

System.out.println("我是有参构造");

this.name = name;

this.age = age;

}

}

## 每种代码块特点:

### 局部代码块:

以”{}”划定的代码区域，此时只需要关注作用域的不同即可

方法和类都是以代码块的方式划定边界的

### 构造代码块

优先于构造方法执行，构造代码块用于执行所有对象均需要的初始化动作

每创建一个对象均会执行一次构造代码块。

### 静态代码块

它优先于主方法执行、优先于构造代码块执行，当以任意形式第一次使用到该类时执行。

该类不管创建多少对象，静态代码块只执行一次。

可用于给静态变量赋值，用来给类进行初始化。

### 案例代码九:

package com.itheima\_04;

/\*

\* Coder静态代码块执行 --- Coder构造代码块执行 --- Coder无参空构造执行

\*

\*

\* BlockTest静态代码块执行 --- BlockTest的主函数执行了 --- Coder静态代码块执行 --- Coder构造代码块执行 --- Coder无参空构造执行

\* Coder构造代码块执行 --- Coder无参空构造执行

\*

\*/

public class BlockTest {

static {

System.out.println("BlockTest静态代码块执行");

}

{

System.out.println("BlockTest构造代码块执行");

}

public BlockTest(){

System.out.println("BlockTest无参构造执行了");

}

public static void main(String[] args) {

System.out.println("BlockTest的主函数执行了");

Coder c = new Coder();

Coder c2 = new Coder();

}

}

class Coder {

static {

System.out.println("Coder静态代码块执行");

}

{

System.out.println("Coder构造代码块执行");

}

public Coder() {

System.out.println("Coder无参空构造执行");

}

}